BX

PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 6 : G06F — A2

(11) International Publication Number: WO 99/32956

(43) International Publication Date: 1 July 1999 (01.07.99)

(21) International Application Number: PCT/US98/25821

(22) International Filing Date: 4 December 1998 (04.12.98)

(30) Priority Data:
08/992,038 19 December 1997 (19.12.97) US

(71) Applicant: HOLONTECH CORPORATION [US/US]; 2039 Samaritan Drive, San Jose, CA 95124 (US).

(72) Inventors: BHASKARAN, Sajit; 1336 Avoset Terrace, Sunnyvale, CA 94087 (US). MATTHEWS, Abraham, R.; 933 Willow Leaf Drive, San Jose, CA 95128 (US).

(74) Agent: MACPHERSON, Alan, H.; Skjerven, Morrill, MacPherson, Franklin & Friel LLP, Suite 700, 25 Metro Drive, San Jose, CA 95110 (US).

(81) Designated States: AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CU, CZ, DE, DK, EE, ES, FI, GB, GE, GH, GM, HU, ID, IL, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: DYNAMIC LOAD BALANCER FOR MULTIPLE NETWORK SERVERS

(57) Abstract

The present invention provides methods and systems for balancing the load on a plurality of servers using a load balancing algorithm which continuously examines the loads on the plurality of servers and makes adjustments in the loads accordingly. Among the factors considered in the load balancing are the power of each server relative to other servers, the load on each server relative to the other servers, and a "credit" for each server based on their power and load.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece			TR	Turkey
BG	Bulgaria	HU	Hungary	ML	Mali	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MN	Mongolia	UA	Ukraine
BR	Brazil	IL	Israel	MR	Mauritania	UG	Uganda
BY	Belarus	IS	Iceland	MW	Malawi	US	United States of America
CA	Canada	IT	Italy	MX	Mexico	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NE	Niger	VN	Viet Nam
CG	Congo	KE	Kenya	NL	Netherlands	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NO	Norway	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	NZ	New Zealand		
CM	Cameroon			PL	Poland		
CN	China	KR	Republic of Korea	PT	Portugal		
CU	Cuba	KZ	Kazakstan	RO	Romania		
CZ	Czech Republic	LC	Saint Lucia	RU	Russian Federation		
DE	Germany	LI	Liechtenstein	SD	Sudan		
DK	Denmark	LK	Sri Lanka	SE	Sweden		
EE	Estonia	LR	Liberia	SG	Singapore		

DYNAMIC LOAD BALANCER FOR MULTIPLE NETWORK SERVERS

BACKGROUND OF THE INVENTION

Field of the Invention

The present invention relates generally to computer networking and, in particular, to a system to perform load balancing on multiple network servers.

Discussion of Related Art

Due to increasing traffic over computer networks such as the Internet, as well as corporate intranets, WANs and LANs, data providers must satisfy an increasing number of data requests. For example, a company that provides a search engine for the Internet may handle over a million hits (i.e. accesses to its web page) every day. A single server cannot handle such a large volume of data requests within an acceptable response time. Therefore, most high-volume information providers use multiple servers to satisfy the large number of data requests.

The prior art does not take into account packets given time interval, and other measures of packet traffic to and from each replicated server. One reason this is not done is because it takes a great number of CPU cycles to compute this on a continual basis for each server that may be connected to a load balancer, which typically is based on a general purpose microprocessor driven by a software program. There is a need for a solution that overcomes this technical hurdle and

incorporates packet traffic loads on the network interfaces belonging to the servers, as packet loss at the server network interface is a frequent reason for throughput and performance degradation.

Figure 1 illustrates a typical arrangement of computers on a network 110. Network 110 represents any networking scheme such as the internet, a local ethernet, or a token ring network. Various clients such as a client 120 or a client 130 are coupled to network 110. Computer equipment from a data provider 160 is also coupled to network 110. For example, data provider 160 may use a bridge 161 to connect a local area network (LAN) 163 to network 110. Servers 162, 164, 166, and 168 are coupled to LAN 163.

Most data transfers are initiated by a client sending a data request. For example, client 120 may request a web page from data provider 160. Typically, on the internet, client 120 requests information by sending a data request to a name such as "www.companyname.com" representing data provider 160. Through the use of the domain name server system of the internet, the name is converted into a IP address for data provider 160. Client 120 then sends a data request to the IP address. The data request contains the IP address of client 120 so that data provider 160 can send the data requested data back to client 120. Data provider 160 converts the IP address into the IP address of server 162, server 164, server 166, or server 168. The data request is then routed to the selected server. The selected server then sends the data requested data to client 120. For other networks, the specific mechanism for routing a data request by client 120 to data provider 160 can vary. However, in most cases data requests from client 120

contain the network address of client 120 so that data provider 160 can send the requested data to client 120.

Since each of the multiple servers contain the same information, each data request can be handled by any one of the servers. Furthermore, the use of multiple servers can be transparent to client 120. Thus the actual mechanism used to route the data request can be determined by data provider 160 without affecting client 120. To maximize the benefits of having multiple servers, data provider 160 should spread the data requests to the servers so that the load on the servers are roughly equal. Thus, most data providers use a load balancer to route the data requests to the servers. As shown in Figure 2, conceptually a load balancer 210 receives multiple data requests 220 and routes individual data requests to server 162, 164, 166, or 168. In Figure 2, four servers are shown for illustrative purposes only. The actual number of servers can vary. Multiple data requests 220, represents the stream of data requests from various clients on network 110.

Some conventional load balancing methods include: random assignment, mod S assignment where S represents the number of servers, and sequential assignment. In random assignment, load balancer 210 selects a server at random for each data request. In modulo-S assignment, each server is assigned a number from 0 to S-1 (where S is the number of servers). Load balancer 210 selects the server which corresponds to the address of the client making the data request modulo S. In sequential assignment, each server is selected in turn for each new data request. Thus, if eight data requests are received, server 162 processes data requests one and five; server 164 processes data requests two and six; server 166

processes data requests three and seven; and server 168 processes data requests four and eight.

On the average over many data requests, each of the conventional load balancing methods should provide adequate load balancing. However, short term imbalances can result from a variety of sources. For example, if some data requests require more data than other data, all three methods may overload one server while other servers are idle. If the majority of data requests come from clients which are mapped to the same server using modulo S assignment, one server becomes overloaded while others servers are underloaded. Thus, conventional load balancing methods do not provide balanced loads in many circumstances. Furthermore, conventional load balancing methods do not adapt to conditions of the servers. Hence there is a need for a load balancer which uses a dynamic load balancing method which can consistently balance the load on a plurality of servers.

SUMMARY OF THE PRESENT INVENTION

The present invention includes methods and systems for consistently balancing the load on a plurality of servers. Specifically, the load balancer uses hashine to separate data requests from clients into a plurality of buckets. The buckets are dynamically assigned to the server having the lightest load as necessary.

The load balancer tracks the state of each server. A server is in the non-operational state if it is deemed as unable to perform the service. The load balancer

maintains a list of services that re operational and assigns load only to servers that are operational.

The server fault tolerance mechanism in the load balancer detects when a server goes down and redistributes the load to the new set of operational servers. Additionally, when a previously non-operational server becomes operations, the traffic is redistributed over the new set of operational servers. This redistribution is done in such a manner as not to disrupt any existing client-server connections.

BRIEF DESCRIPTION OF THE DRAWINGS

Figure 1 is a representational showing of a prior art network connecting a plurality of servers to a plurality of clients;

Figure 2 illustrates a prior art load balancer connecting a network to a plurality of servers;

Figure 3 illustrates how a load balance in accordance with this invention determines to which server multiple data request are allocated;

Figure 4 is a block diagram of a load balance in accordance with one embodiment of this invention.

Figure 5a and 5b illustrate multi-mode buckets used in some embodiments of the invention;

Figure 6a - 6d illustrate the effect of a new bucket assignment;

Figure 7, 8 and 9 are simplified flow charts summarizing the action performed by the load balancer of this invention;

Figure 10 is a block diagram illustrating the operation of one embodiment of the server fault detection of this invention; and

Figure 11 illustrates a load balancer in the form of a flow switch and in which each a plurality of servers has the same IP address.

DETAILED DESCRIPTION

According to the principles of this invention, certain limitations imposed by conventional load balancers have been overcome. The present invention provides a load balancer that monitors the load on each server to determine how to allocate data requests among the servers. Specifically, the load balancer separates the data requests into buckets and assigns buckets to the least loaded server.

Figure 3 illustrates conceptually how a load balancer in accordance with one embodiment of the invention determines to which server each data request is allocated. Multiple data requests 220 are separated into a first plurality of buckets 310. As used herein, the number of buckets is B and the bucket in first plurality of buckets 310 are numbered from 310_0 to 310_(B-1). Each bucket is assigned to one server in plurality of servers 320 as explained below. As used herein, the number of servers is equal to S and the individual servers are numbered 320_0 to 320_(S-1). As explained in detail below, a single bucket can be separated into a plurality of child buckets.

To increase flexibility, the number of buckets B should be user configurable. Generally, a larger number of buckets B will result in a more balanced load on the servers. However, if the address space of the clients making data requests are large; a smaller number of buckets B is acceptable. In one

specific embodiment of a load balancer in accordance with the present invention, the number of buckets B is configurable to be 64 or 128 or 256.

In one embodiment of the present invention, data requests are assigned to buckets using Modulo-B assignment based on the address of the client making the data request. Thus, in this embodiment, a data request is assigned to bucket 310_((Client_Address) MOD B). Therefore, data requests from different clients may share the same bucket. One benefit of using Modulo-B assignment is to cause all data requests from a single client to be assigned to the same bucket in plurality of buckets 310. Since each bucket is assigned to only one server, all data requests from a single client are directed to a single server. If having all of a client's data requests directed to a single server is not important, other methods such as random assignment or sequential assignment can be used to assign data requests to plurality of buckets 310.

Figure 4 is a block diagram of a load balancer 400 in accordance with one embodiment of the present invention. In Figure 4, a bucket selector 410 assigns data requests in multiple data requests 220 to buckets of first plurality of buckets 310 (Figure 3). A server selector 420 assigns buckets from first plurality of buckets 310 as well as other buckets created by a bucket controller 430 (as explained below) to servers in plurality of servers 320. Server selector 420 assigns a bucket to a server based on data from credit calculator 450, bucket controller 430, and bucket selector 410. Server selector 420 assigns unassigned buckets to the server with the largest credit as determined by credit calculator 450.

Load estimator 460 monitors various load statistics about the servers to determine the load on each server. Credit calculator 450 determines the amount of free resources or credit each server has remaining from the load estimates provided by load estimator 460. Skew detector 440 uses the credit of each server to determine whether the load balancing between the servers is in a tolerable range. If the load is skewed, i.e. outside the tolerable range, skew detector 440 signals bucket controller 430 to create additional buckets as explained below.

Figures 5(a) and 5(b) illustrate multi-mode buckets used in some embodiments of the present invention. Specifically, buckets can be in low granularity mode or high granularity mode. Buckets in first plurality of buckets 310 begin in low granularity mode. In certain situations (as described below), a bucket in first plurality of buckets 310, such as bucket 310_0, is placed into high granularity mode by bucket controller 430. In high granularity mode, new data requests in bucket 310_0 are separated into a plurality of child buckets 520. Child buckets are created to be in low granularity mode. Assignment of data requests to bucket 310_0 to plurality of child buckets 520 is done by straight selection. In one embodiment, plurality of child buckets 520 contains one child bucket for each client having pending data requests in bucket 310_0. If additional data requests from different clients are sent to bucket 310_0, bucket controller 430 creates additional child buckets in plurality of child buckets 520. Bucket controller 430 may place other buckets in first plurality of buckets 310 into high granularity mode to create additional pluralities of child buckets.

In some embodiments of the present invention, buckets are assigned to one of several states to track whether a bucket is in low or high granularity mode. The bucket states for one embodiment include: NOT ASSIGNED, ASSIGNED LOW, ASSIGNED TRANSITION HIGH, ASSIGNED HIGH, and ASSIGNED TRANSITION LOW.

Buckets, which are not yet assigned to a server are in the NOT ASSIGNED state. First plurality buckets when assigned to a server are in the ASSIGNED LOW state.

Buckets, which are in high granularity mode and already assigned to a server are in the ASSIGNED HIGH state.

When bucket controller changes the mode of a bucket from low granularity to high granularity, the bucket is in the ASSIGNED TRANSITION HIGH state. When in this state, additional data requests from different clients that do not correspond to a child bucket create additional child buckets. All the child buckets are assigned to the same server as the first plurality bucket.

When the bucket completes the transition to high granularity, the state of the bucket changes to the ASSIGNED HIGH state. In this state the first plurality of buckets are no longer assigned to a server. Data requests from clients that do not correspond to a child bucket create additional child buckets. In this state, different child buckets are assigned to different servers.

When bucket controller 430 changes the mode of a bucket from high granularity to low granularity, the bucket is in the ASSIGNED TRANSITION LOW state. In this state, the first plurality of buckets are assigned to a server. Data

requests from clients that do not correspond to a child bucket do not create additional child buckets. These data requests are sent to the server assigned to the first plurality bucket.

When the bucket completes the transition to low granularity mode, the state of the bucket changes to the ASSIGNED LOW state.

As explained below, the buckets are used to forward data requests from clients to servers. If a bucket does not receive a data request in some fixed time, i.e. the time-out period, the bucket transitions to the NOT ASSIGNED state

As explained above, as data requests are received by load balancer 400, bucket selector 410 assigns the data requests to first plurality of buckets 310 (Figure 3). If a bucket in first plurality of buckets 310 is in high granularity mode, the data requests are further divided into a plurality of child buckets belonging to the bucket in first plurality of buckets 310 (Figure 3). Server selector 420 (Figure 4) assigns unassigned buckets which have active data requests to a server in plurality of servers 320 (Figure 3).

Specifically, whenever an unassigned bucket receives a data request, server selector 420 assigns that bucket to the server with the most credit as calculated by credit calculator 450. In embodiments using states, the state of the bucket is changed from NOT ASSIGNED to ASSIGNED LOW. Once a bucket is assigned to a server, the bucket remains assigned to that server until the bucket times out due to lack of data requests, which causes the bucket to change to the NOT ASSIGNED state, or until bucket controller 430 forces to bucket to high granularity mode.

If bucket controller 430 forces a bucket, for example bucket 310_0 (Figure 3), to high granularity mode, the state of the bucket is changed to ASSIGNED TRANSITION HIGH. Bucket controller 430 then creates a plurality of child buckets 520, each of which is in the ASSIGNED LOW state. Some network data requests, such as TCP data requests, can not be reassigned to a different server. Therefore, one embodiment of load balancer 400 does not reassign any of the pending data requests. However, since some network data requests, such as UDP data requests, are reassignable, another embodiment of load balancer 400 reassigns pending data requests which can be reassigned to child buckets.

As new data requests are routed to bucket 310_0 by bucket selector 410; bucket controller 430 routes the data requests to the appropriate child bucket. If an appropriate child bucket is not found and a new child bucket is created, server selector 420 assigns the child bucket to the server with the most credit. The state of the child bucket then transitions to the ASSIGNED LOW state. Thus, each unassigned buckets which has a data request is assigned by server selector 420 to the operational server which currently has the most credit.

Operational servers are assigned credits by the credit calculator 450. Credit calculator 450 determines the credit of each operational server based on the relative power of each server and the load on each server. Credit calculator 450 determines which servers are underloaded relative to the power of each operational server. Underloaded servers have loads which are smaller than overloaded servers relative to the power of each server. In some embodiments, credit calculator 450 receives load information directly from each server. In other embodiments, the load on an

operational server is determined by load estimator 460 as explained below. Generally, the load on each server is reduced to a scalar number. If the servers in plurality of server 320 are not of equal power, credit calculator 450 should be provided with a relative power rating for each server. In one embodiment of credit calculator 450, the relative power rating of each server is provided as a scalar number.

Method Description

Integer computations are faster in a computer than floating point computations. Thus all calculations are scaled appropriately to employ only integer calculations at the expense of a certain loss of computational precision. Two different scaling factors are used - namely *WeightScale* and *LoadScale*. In one embodiment the both scaling factors used are 1000.

Specifically, the weights, $\overline{Wi}$, of each server i is computed by dividing each power rating by the sum of the power ratings, as shown in equation 1:

$$\overline{Wi} = WeightScale * Pi / \left(\sum_{n=0}^{S-1} Pn \right) \qquad (1)$$

where Pi is the power rating of each server i (numbered from 0 to S-1) and $\overline{Wi}$ is the weight of each server. The sum of the weights of every server is equal to *WeightScale* and each weight is in the range of one to *WeightScale*. The weights are computed every time the set of operational servers changes.

Specifically, the credit calculator 450 at periodic intervals, calculates a weighted load for each server as shown in equation 2:

$$\overline{\mathrm{Li}} = LoadScale * \mathrm{Li}/\overline{Wi} \quad (2)$$

where Li is the load on server i (numbered from 0 to S-1) and $\overline{\mathrm{Li}}$ is the weighted load on server i.

The load normalizer is calculated from $\overline{\mathrm{Li}}$ as shown in equation 3:

$$LN = \min_{i=0}^{S-1} abs(\overline{\mathrm{Li}}) \quad (3)$$

Specifically, the credit calculator 450 calculates the credit of each server, Ki, is calculated by subtracting the normalized weighted load of the server from the weight of the server, as shown in equation 4:

$$\mathrm{Ki} = \overline{\mathrm{Wi}} - (\overline{\mathrm{Li}} / LN) \quad (4)$$

A flow weight is calculated from Ki as shown in equation 5:

$$\mathrm{Fw} = \min_{i=0}^{S-1} abs(\mathrm{Ki}) \quad (5)$$

Servers that are overloaded relative to their power ratings may have negative credits and servers that are underloaded relative to their power rating may have positive credits. The server that is the most underloaded has the largest credit.

Instead of recalculating the credit of every server after a bucket is assigned, some embodiments of load balancer 400 reduce the computational overhead of credit calculator 450 by approximating the effect of a new bucket assignment. When server selector 420 assigns a bucket to a server, it calculates a flow adjustment for said server shown in equation 6:

$$Fa = LoadScale * \mathrm{Fw} / \overline{\mathrm{Wi}} \tag{6}$$

Specifically, when server selector 420 assigns a bucket to a server, it adjusts the credit of the said server as shown in equation 7:

$$\mathrm{Ki} = \mathrm{Ki} - Fa \tag{7}$$

Figure 6(a)-6(d) illustrates the computation performed. Buckets in plurality of buckets 610 are already assigned to server 660, server 670 and server 680. Buckets 620, 630, and 640 are unassigned and will be in the NOT ASSIGNED state in embodiments using bucket states. The power rating of server 660 is 10; the power rating of server 670 is 20 and the power rating of server 680 is 30. The algorithm implicitly normalizes all quantities to an idealized server and the effective load that the idealized server experiences. In the following discussion, the result of calculations in Table 1 is compared with an idealized server. The idealized server has 1000 units of capacity.

The weights (or relative capacity) for servers 660, 670 and 680 are 167, 333, and 500 respectively (and sum up to the idealized servers capacity)

Table 1 shows the computational results at 2 time epochs. All computations are done with integer arithmetic. This ensures that the computations are fast. To avoid loss of precision due to integer computations a scaling factor is used.

5

Table 1: Motivation for Load Balancing Algorithm

		Ideal	*Server 660*	*Server 670*	*Server 680*
1	Power rating		10	20	30
2	Server Weight	1000	167	333	500
3	Load at time *t*		100	100	100
4	Weighted load at time *t*	110	60 = (100 * 100)/167	30 = (100 * 100)/333	20 = (100 * 100)/500
5	Load normalizer at time *t*	20			
6	Server credits at time *t*	995	164 = 167 - 60/20	332 = 333 - 30/20	499 = 500 - 20/20
7	Flow weight at time *t*	164			
8	Flow adjustment at time *t+k*	164	98 = 100 * 164)/167	49 = 100 * 164)/333	32 = (100 * 164)/500
9	Load at time *l*		300	200	50
10	Weighted load at time *l*	249	179 = (100 * 300)/167	60 = (100 * 200)/333	10 = (100 * 50)/500
11	Load normalizer at time *l*	10			
12	Server credits at time *l*	973	150 = 167 – 179/10	324 = 333 – 90/10	499 = 500 – 10/10
13	Flow weight at time *l*	150			
14	Flow adjustment at time *l+m*	150	89 = (100 * 150)/167	45 = (100 * 150)/333	30 = (100 * 150)/500

In Table 1, rows 1 and 2 are computed by credit calculator 450 every time the set of operational servers changes.

Row 1: User defined power ratings (P_i).

Row 2: The weights (or relative computational capacity) for each server (W_i) in accordance with equation 1.

At time epoch *t*, load estimator 460 gets the load vector and converts it into a scalar quantity L_i. Since the servers are of different computing capacity, load

estimator 460 normalizes the load before they can be compared. Intuitively, a load of 100 units to a server of capacity 10 should equal a load of 200 units to a server of capacity 20. Thus load estimator 460 normalizes based on the inverse of the server weights.

Server credits are computed periodically to decide which server has the maximum capacity available. In Table 1, rows 3 to 7, show the computation of server credits by credit calculator 450 at epoch t when the raw loads are all equal.

Row 3: Current load per server (L_i). This is computed from a load vector.

Row 4: The weighted load for each server is computed in accordance with equation 2. Note that even though each server has the same load, it represents a larger load for server 660 than for server 670 or server 680. The effective load for idealized server is the sum of the normalized load for all servers.

Row 5: To relate the load to capacity, the load normalizer LN is computed in accordance with equation 3. Evidently this value has to be a function of the current load.

Row 6: A normalized estimate of computational resources used by the current load is subtracted from the servers weight to arrive at the servers credits (K_I) in accordance with equation 4.

Note that the idealized server credits can be computed in 2 different ways. In one method, we use the same formula that we use for each server. Alternately we could sum up the credits of all servers. Both methods should come out to be equal (or very nearly equal due to integer arithmetic).

Row 7: Finally an estimate of computational use by a new flow is needed for each server. Towards this goal we compute the flow weight in accordance with equation 5. This is also the amount that we must deduct from the idealized servers credits when a flow is assigned to it.

Whenever a flow is assigned to a server at a later time epoch (say $t+k$), the computational resource to be subtracted from the server credits is shown in Table 1, rows 8 (in accordance with equation 6). This credit adjustment is performed by server selector 420 whenever a bucket to server assignment occurs.

Row 8: The computational resource that a flow consumes for each server is computed in accordance with equation 6. This is the value to deduct from the servers credits when a new bucket is assigned to the server.

Note that we can assign 3 flows to server 680, 2 flows to server 670 and 1 flow to server 660 and each individual server credits would be diminished by the same amount.

In Table 1, rows 9 to 14, show the computation at another epoch when the raw loads are not equal.

Since the bucket debit amount is only an approximation of the load added to a server by a bucket, credit calculator 450 must periodically recalculate the credit of every server using load information of load estimator 460. For a typical internet web server business, recalculation of the server credits should be performed every ten to twenty seconds. Recalculation should be performed more often if a large number of data requests are received.

As explained above, load estimator 460 provides credit calculator 450 with an estimate of the load on each server. Common measures of the load on a server are based on the number of data requests, the number of data packets received by the server, the number of bytes received by the server, the number of data packets sent by the server, the number of bytes sent by the server, the processor activity on the server, and the memory utilization on the server. Because load balancer 400 is primarily designed to balance the network load on a plurality of servers, load estimator 460 typically uses load measures based on network traffic to the server. Thus most embodiments of load estimator 460 use load measures based on the number of data requests received by the server, the number of data packets received by the server, the number of bytes received by the server, the number of data packets sent by the server, or the number of bytes sent by the server.

One embodiment of the load estimator 460 uses a combination of the number of data packets received by the server, the number of bytes received by the server, the number of data packets sent by the server, and the number of bytes sent by the server. This embodiment of load estimator 460 forms a load vector with four scalar quantities corresponding to the number of data packets received by the server, the number of bytes received by the server, the number of data packets sent by the server, and the number of bytes sent by the server. This embodiment of load estimator 460 calculates the load of a server using the cross product of the load vector with a weighting vector, having four scalar weights. Thus, the load of a server is calculated as shown in equation 8:

$$L = < P\_in, B\_in, P\_out, B\_out > \otimes < W\_pin, W\_bin, W\_pout, W\_bout >$$
$$= P\_in * W\_pin + B\_in * W\_bin + P\_out * W\_pout + B\_out * W\_bout \quad (8)$$

where P_in is the number of packets received by the server, B_in is the number of bytes received by the server, P_out is the number of packets sent by the server, B_out is the number of bytes sent by the server, W_pin is the weight assigned to the number of packets received by the server, W_bin is the weight assigned to the number of bytes received by the server, W_pout is the weight assigned to the number of packets sent by the server, and W_bout is the weight assigned to the number of bytes sent by the server. In most embodiments, the weight vector is provided by the user of load balancer 400. However, some embodiments hard code the weight vector in firmware, software, or hardware.

As explained above, bucket controller 430 converts buckets from low granularity mode to high granularity mode to balance the load on the servers. Bucket controller 430 works in conjunction with skew detector 440. Skew detector 440 determines when the load on the servers are skewed, i.e. unbalanced. Specifically, skew detector 440 compares a measure of skewness against a skew threshold. If the absolute value of measure of skewness is greater than the skew threshold, skew detector 440 causes bucket controller 430 to change one or more buckets from low granularity mode to high granularity mode.

To keep the measure of skew independent of the design of the load estimator 460, only normalized measures of skew are used. Thus, in one

embodiment the measure of skewness used by skew detector 440 is given in equation 7. In another embodiment the measure of skewness may be computed by equation 8.

$$\overline{Li} = Li / \left(\sum_{n=0}^{S-1} Ln \right) \quad (7)$$

$$\overline{Li} = Li / \left(\sum_{n=0}^{S-1} (Ln)^2 \right) \quad (8)$$

If skew detector 440 directs bucket controller 430 to convert one or more buckets from low granularity mode to high granularity mode, bucket controller 430 must determine which bucket to convert. In one embodiment, exactly one bucket is selected for transition from low to high granularity state. In this embodiment one bucket is selected at random from the set of buckets assigned to the server with the largest load.

In another embodiment, one or more buckets are selected for transition from low to high granularity state. For each server that has a skewed load, 1 bucket is selected at random from the set of buckets assigned to that server. In still another embodiment, a fuzzy logic controller with a set of rules for bucket selection is used.

The transition of a bucket from low granularity mode to high granularity mode is not instantaneous. As explained above, in some embodiments the bucket first transitions to the ASSIGNED TRANSITION HIGH state before reaching the ASSIGNED HIGH state. After a configurable aount of time has transpired, the bucket controller 430 forces the bucket in high granularity state to low granularity

state. Therefore, in most embodiments of load balancer 400, bucket controller 430 only converts one bucket from low granularity mode to high granularity mode when skew detector 440 detects a skewed load. Skew detector 440 should then wait for a time before rechecking for a skewed load. However, some embodiments of load balancer 400 may cause bucket controller 430 to convert a plurality of buckets every time skew detector 440 detects a skewed load.

Figures 7, 8, and 9 are three simplified flow charts which summarize the actions performed by load balancer 400. The processes of the three flow charts of Figures 7, 8, and 9 are performed simultaneously in load balancer 400. Specifically, Figure 7 summarizes the flow of a data request through load balancer 400. In WAIT FOR DATA REQUEST 710, load balancer 400 waits for data requests from network 110. When a data request is received, the data request is assigned to a bucket in ASSIGN REQUEST TO DATA BUCKET 720. Once the data request is assigned to a bucket, load balancer 400, or more specifically bucket controller 430 (Figure 4), must determine if the data request must be redirected to a child bucket in REDIRECT TO CHILD BUCKET? 730. If the data request must be directed, the data request is directed to a child bucket in ASSIGN TO CHILD BUCKET 740.

Once the data request is properly assigned to a bucket or child bucket, the data request is sent to the server to which the bucket or child bucket is assigned in SEND REQUEST TO BUCKET 750. SEND REQUEST TO BUCKET 750 may be delayed if the bucket is not yet assigned. The bucket is assigned by the process described by the flow chart of Figure 8. Load balancer 400 then returns to WAIT

FOR DATA REQUEST 710. Although the flow chart of Figure 7 shows a sequential processing of data requests, data requests are actually pipelined in load balancer 400 since bucket selector 410 can assign data requests to buckets simultaneously as server selector 420 assigns buckets to servers and bucket controller 430 controls buckets.

Figure 8 summarizes the process of assigning buckets to servers for server selector 420. In LOAD NEW CREDIT ALLOCATION 810, load server selector 420 must determine whether to load a new credit allocation from credit calculator 450. IF a new credit allocation is required, server selector 420 loads the credit allocation in LOAD CREDIT ALLOCATION 820. In either case, server selector 420 then detects if any unassigned bucket, i.e. buckets in the NOT ASSIGNED state, contain data requests in DETECT UNASSIGNED BUCKETS WITH REQUESTS 830. If no unassigned buckets contains data requests, server selector 420 returns to LOAD NEW CREDIT ALLOCATION? 810. Otherwise, server selector 420 assigns the unassigned bucket containing a data request to the server with the most credit in ASSIGN BUCKET 840. For embodiments of load balancer 400 using bucket debit amount approximation as described above, server selector 420 subtracts the bucket debit amount from the credit of the server receiving the bucket assignment in DEBIT CREDIT 850. Then server selector 420 returns to LOAD NEW CREDIT ALLOCATION? 810. For embodiments not using bucket debit amount approximation, server selector returns to LOAD NEW CREDIT ALLOCATION? 810 directly from ASSIGN BUCKET 840.

Figure 9 summarizes the interaction of load estimator 460, credit calculator 450, skew detector 440, and bucket controller 430 in load balancer 400.. In ESTIMATE LOAD 910, load estimator 460 estimates the load of the servers as described above. Credit calculator 450 uses the load estimates from load estimator 460 to calculate the credit of each server in CALCULATE CREDIT 920. Skew detector 440 then detects whether the load on the servers is skewed in DETECT SKEW 930. If the load on the servers is not skewed, load balancer 400 returns to ESTIMATE LOAD 910. If the load on the servers is skewed, bucket controller 430 selects a bucket in SELECT BUCKET 940 and causes the bucket to transition from low granularity mode to high granularity mode in TRANSITION BUCKET 950. Load balancer 400 then returns to ESTIMATE LOAD 910.

In addition to the three process illustrated in Figures 7, 8, and 9, buckets which are assigned can become unassigned, i.e. transition to the NOT ASSIGNED state, if the buckets do not receive data requests after a time-out period.

Some embodiments of load balancer provide for a administratively controlled graceful take down of a selected server from the plurality of servers. In this method, the bucket controller 430 forces all buckets currently assigned to said server to ASSIGNED TRANSITION HIGH state for a specified period of time. After the said time has transpired, the bucket controller 430 forces the said bucket to the ASSIGNED TRANSITION LOW state. In this way, traffic assigned to said server is reassigned without disrupting client-to-server communication.

Some embodiments of load balancer 400 are enhanced to handle server faults, i.e. a server developing a problem which causes the server to not respond or

to respond poorly to data requests. As shown in Figure 10, a server fault detector 1010, which detects server faults, is included in a load balancer 1000. In some embodiment server fault detector 1010 and load estimator 460 are combined.

Server fault detector 1010 can be configured to detect server faults in a variety of ways including many well known fault detection routines. For example, server fault detector 1010 could "ping" each server periodically to determine if the server is up. Server fault detector 1010 can also poll the servers periodically to request the status of each server. In one embodiment of load balancer 1000, in addition to polling the servers, server fault detector 1010 analyzes the number of incoming and outgoing packets of a server to determine if the server has a server fault. For example, if a server receives many incoming packets, which most likely are data requests, and has little or no outgoing packets, the server is most likely experiencing a server fault.

Some embodiments of server fault detector 1010 are configured to detect link faults as well as server faults. Since link faults may completely cut off one or more servers from load balancer 1000, link faults are equally damaging as server faults. In one embodiment, the hardware can detect link faults.

Once server fault detector 1010 determines that a server is down, i.e. the server has a server fault or a link fault, server fault detector 1010 marks the server as inactive and removes it from the set of operational servers. Server fault detector 1010 then informs credit calculator 450 about the change in set of operational servers. Credit calculator 450 then recomputes each servers normalized weight and

credits. Credit calculator will ignore inactive servers in performing calculations. The Server selector ignores inactive servers when performing bucket assignments.

Server fault detector 1010 also informs bucket controller 430 of the identity of a down server. In one embodiment of load balancer 1000, bucket controller 430 deassigns all buckets assigned to the down server. Thus the state of the buckets previously assigned to the down server transition to the NOT ASSIGNED state. Server selector 420 can then reassign the buckets previously assigned to the down server to the other servers as described above. In another embodiment of server 1000, bucket controller 1000 reassigns all the buckets assigned to the down server to another server, typically the server with the most credit. Load balancer 1000 rebalances the load among the remaining servers as described above.

Server fault detector 1010 should continue to monitor all inactive servers, to detect when an inactive server may be repaired. Some embodiments of server fault detector 1010, tests a repaired server to assure that the server is actually repaired. Once the down server is repaired to the satisfaction of server fault detector 1010, server fault detector 1010 marks the server as active and adds it to the set of operational servers. Server fault detector 1010 then informs credit calculator 450 about the change in operational servers. Credit calculator 450 then recomputes each servers normalized weight and credits. Since no buckets are assigned to the previously down server, the previously down server should have no load and, therefore, a large credit. Consequently, server selector 420 assigns unassigned buckets with data requests to the previously down server as described above. In time load balancer 400 is able to rebalance the load on all of the servers.

To hasten the load balancing process, skew detector 440 may cause bucket controller 430 to move one or more buckets from low granularity mode to high granularity mode as described above.

Fig. 11 illustrates an overall system in which the present invention is utilized. Fig. 11 shows a plurality of servers 210, 220, 230, 240, 250 each having the same IP address. Servers are connected to a plurality of network routers 260, 270, 280 through a network flow switch 205 in accordance with this invention. The structure and method of operation of the flow switch one disclosed in the above copending application serial no. 08/994,709 (M-4968) which is incorporated herein by reference in its entirety.

In the various embodiments of this invention, methods and structures have been described that provide dynamic load balancing and server fault tolerance for a plurality of servers. By monitoring the load on the servers and dynamically allocating buckets based on the credit of each server, embodiments of the present invention can maintain a balanced load on the servers. Furthermore, by monitoring the status of the servers, embodiments of the present invention can gracefully handle server faults by balancing the load of a down server among the remaining operational servers.

The various embodiments of the structures and methods of this invention that are described above are illustrative only of the principles of this invention and are not intended to limit the scope of the invention to the particular embodiments described. In view of this disclosure, those skilled-in-the-art can define other bucket selectors, server selectors, skew detectors, credit calculators, load

estimators, bucket controllers, server fault detectors, load vectors, weighting vectors, credit allocation rules, load estimation rules, fault detection rules, buckets, bucket states, network configurations, and use these alternative features to create a method, circuit, or system according to the principles of this invention.

CLAIMS

We claim:

1. A method for assigning a plurality of data requests among a plurality of servers, said method comprising:

assigning each of said data requests to a bucket of a first plurality of buckets; and

assigning each bucket of said plurality of buckets containing a data request to a server of said plurality of servers;

wherein said first plurality of buckets outnumber said plurality of servers.

2. The method of Claim 1, wherein each of said data requests includes a source address and wherein each of said data requests is assigned to a bucket based on said source address.

3. The method of claim 2, wherein

said first plurality of buckets contain B buckets;

each of said buckets in said first plurality of buckets is uniquely numbered from zero to B-1; and

a data request of said plurality of data requests is assigned to bucket number said source address of said data request modulo B.

4. The method of claim 1, wherein said assigning each bucket of said plurality of buckets containing a data request to a server of said plurality of servers further comprises:

detecting a first not-assigned bucket having at least one of said data requests;

assigning said first not-assigned bucket to an underloaded server.

5. The method of claim 4, wherein said underloaded server is the most underloaded server.

6. The method of claim 1, wherein assigning each bucket of said plurality of buckets containing a data request to a server of said plurality of servers further comprises:

detecting a first not-assigned bucket having at least one of said data requests;

calculating a weighted load on each of said server based on said server power;

calculating a credit for each of said servers;

assigning said first not-assigned bucket to a server having the largest credit.

7. The method of claim 6, further comprising detecting a skewed load on said plurality of servers.

8. The method of claim 7, further comprising:

selecting a second bucket from said plurality of buckets, wherein said second bucket is in a low granularity mode; and

converting said second bucket from said low granularity mode to a high granularity mode.

9. The method of claim 8 further comprising:

creating a plurality of child buckets; and

assigning data requests destined for said second bucket to one of said plurality of child buckets.

10. A load balancer for balancing the load due to a plurality of data requests on a plurality of servers, said load balancer comprising:

a first plurality of buckets;

a bucket selector configured to assign each of said data requests to a bucket in said plurality of buckets; and

a server selector configured to dynamically assign any unassigned bucket containing a data request to a server in said plurality of servers.

11. The load balancer of claim 11, wherein said plurality of buckets outnumbers said plurality of servers.

12. The load balancer of claim 10, wherein

each of said data requests contains a source address; and

said bucket selector determines which bucket to assign a specific data request based on said source address of said specific data request.

13. The load balancer of claim 10, further comprising a load estimator configured to estimate a load on each of said servers to form a plurality of loads.

14. The load balancer of claim 13 wherein said server selector assigns an unassigned bucket containing a data request to an underloaded server.

15. The load balancer of claim 14, wherein said underloaded server is a most underloaded server.

16. The load balancer of claim 13, further comprising a credit calculator coupled to said load estimator and configured to calculate a credit for each of said servers to form a plurality of credits.

17. The load balancer of claim 13, wherein said server selector assigns an unassigned bucket containing a data request to a server having a large credit.

18. The load balancer of claim 17, wherein said server selector assigns an unassigned bucket containing a data request to a server having the largest credit.

19. A server fault handling system for handling the non-operational status of one of a plurality of servers, said server receiving requests in buckets; comprising,

fault detection means for detecting the non-operational status of said server,

credit calculation means responsive to said server fault detection means for recalculating parameters of said plurality of servers based on the loss of said non-operational server,

20. A fault handling system in accordance with claim 19 in which said server selection means reassigns all of said buckets from non-operational server to the one of said plurality of servers having the lightest load.

21. A fault handling system in accordance with claim 19 in which said server selection means reassigns said buckets for said non-operational server to said other servers.

22. A server fault handling system in accordance with claim 19 including

means for notifying said server fault detector that said formerly non-operational server is now operational,

means for notifying said credit calculator of the operational status of said formerly non-operational server to cause said credit calculator to increase the credit of said now operational server, and

means in said selector means responsive to said increased credit of said now operational server for assigning unassigned buckets to said now operational server.

23. A fault handling system in accordance with claim 22 in which said means in said server selection means reassigns assigned buckets to all of said plurality of servers including said now operational server.

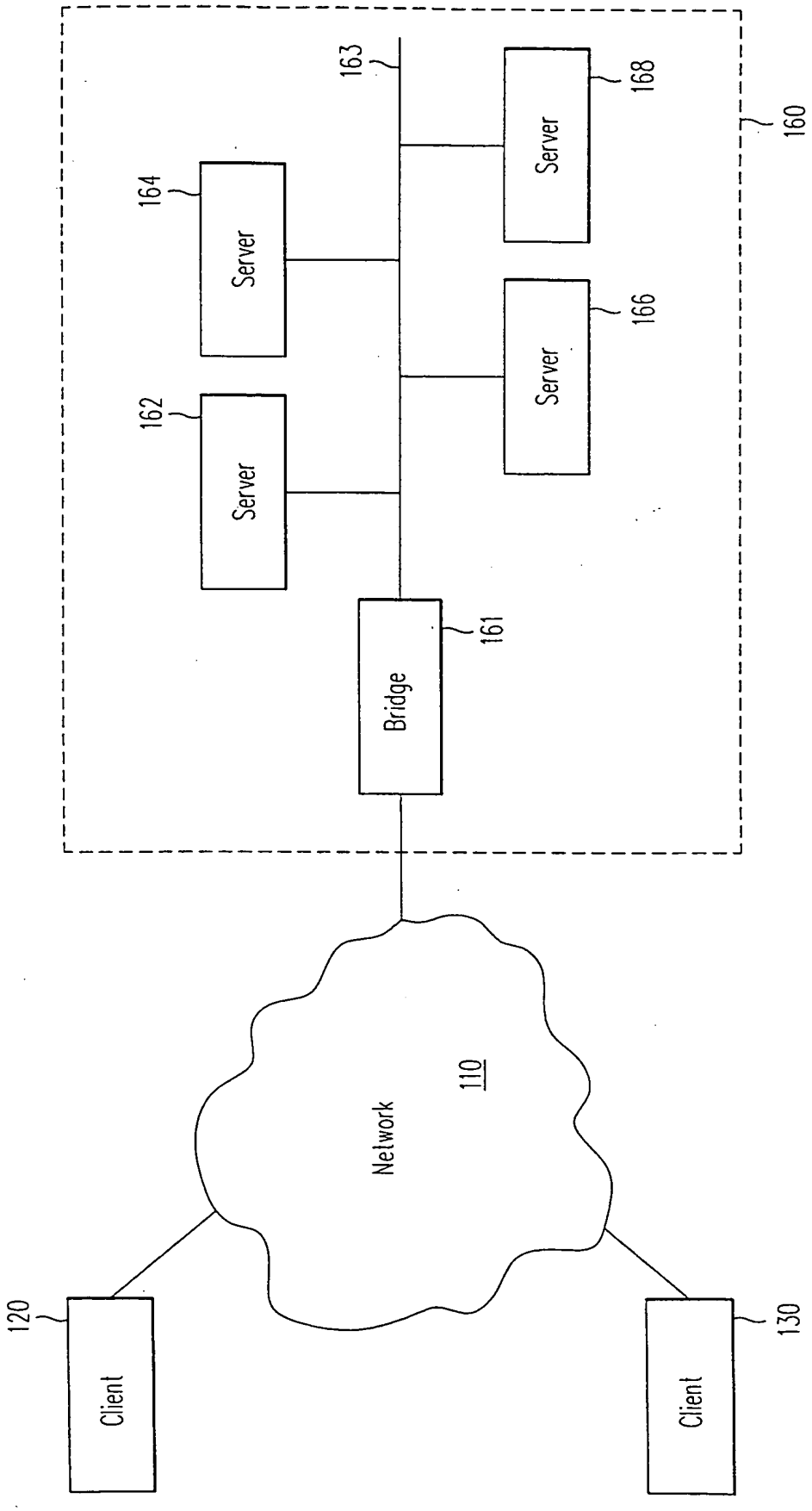

FIG. 1
(Prior Art)

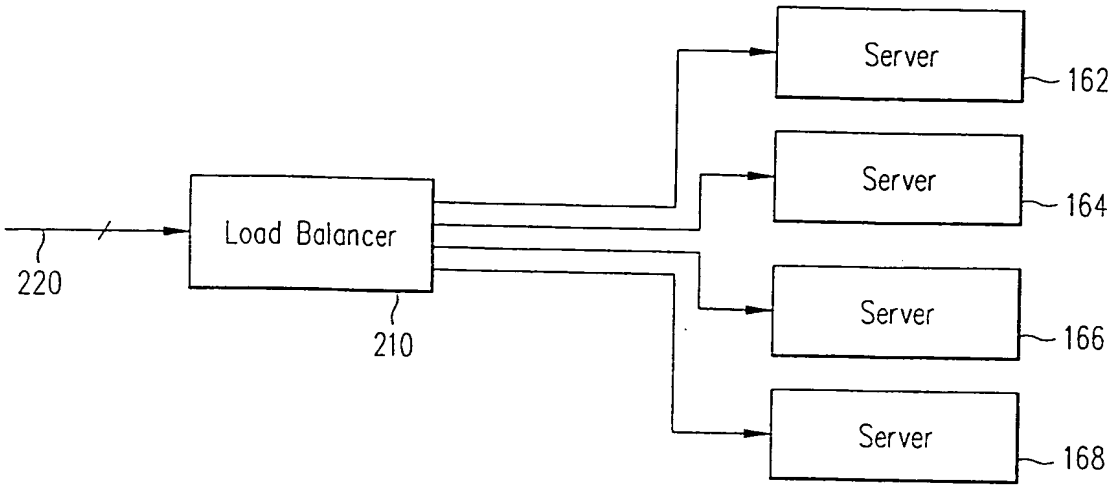

FIG. 2
(Prior Art)

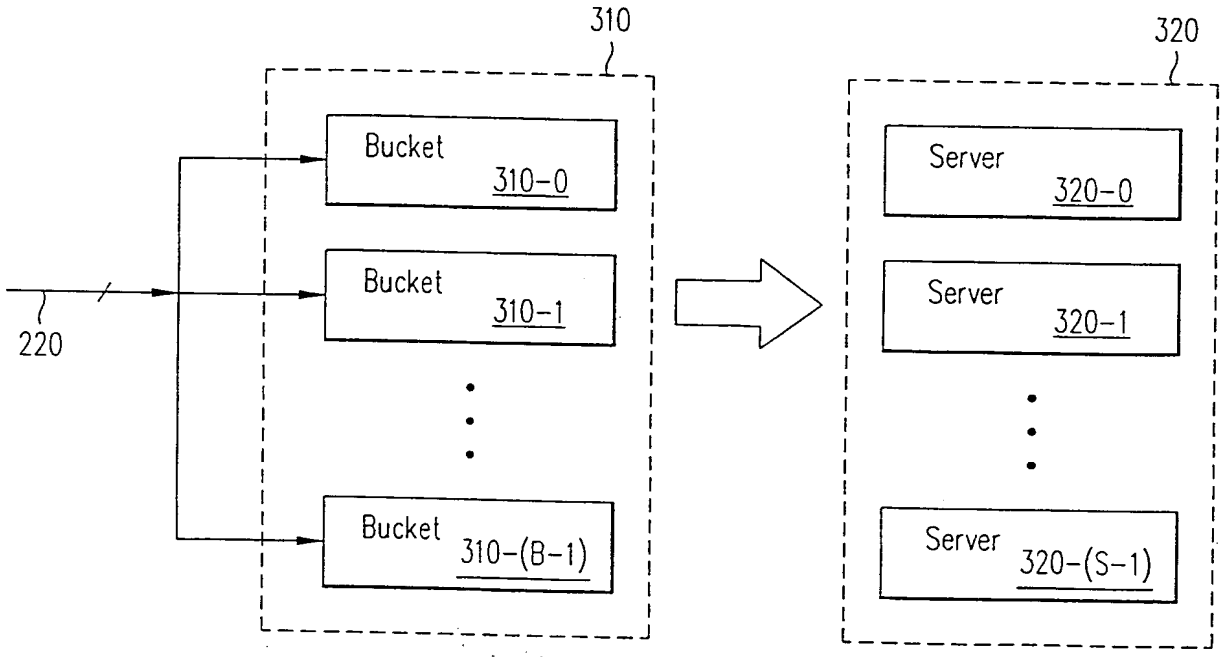

FIG. 3

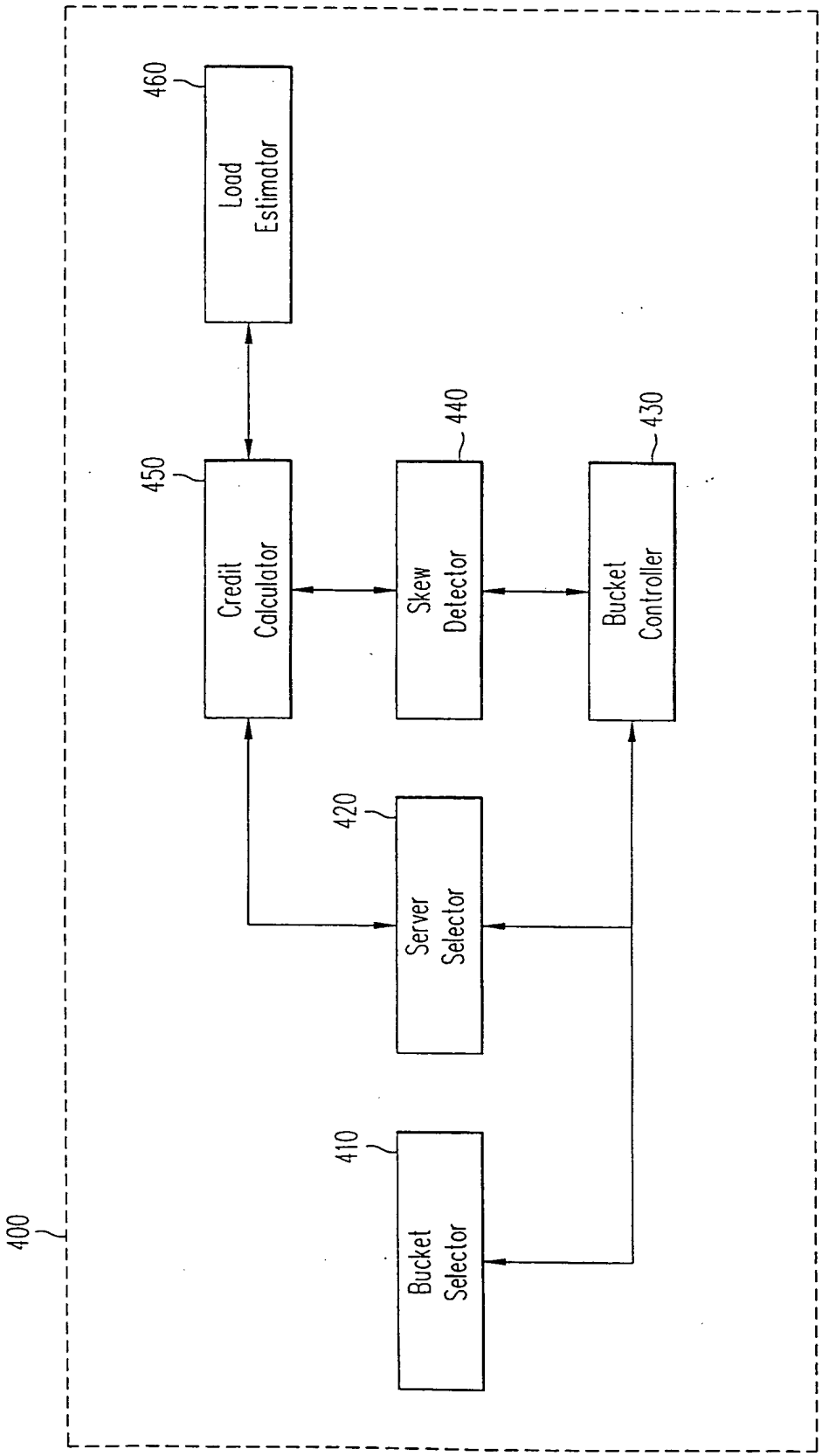
400
460
Load Estimator
450
Credit Calculator
440
Skew Detector
430
Bucket Controller
420
Server Selector
410
Bucket Selector

FIG. 4

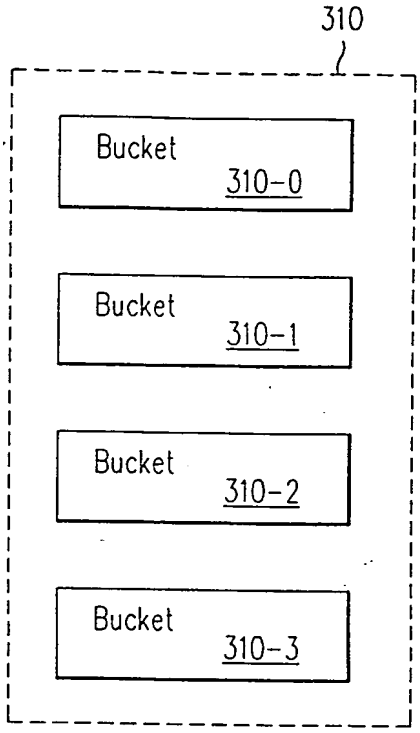
310
Bucket 310-0
Bucket 310-1
Bucket 310-2
Bucket 310-3

FIG. 5A

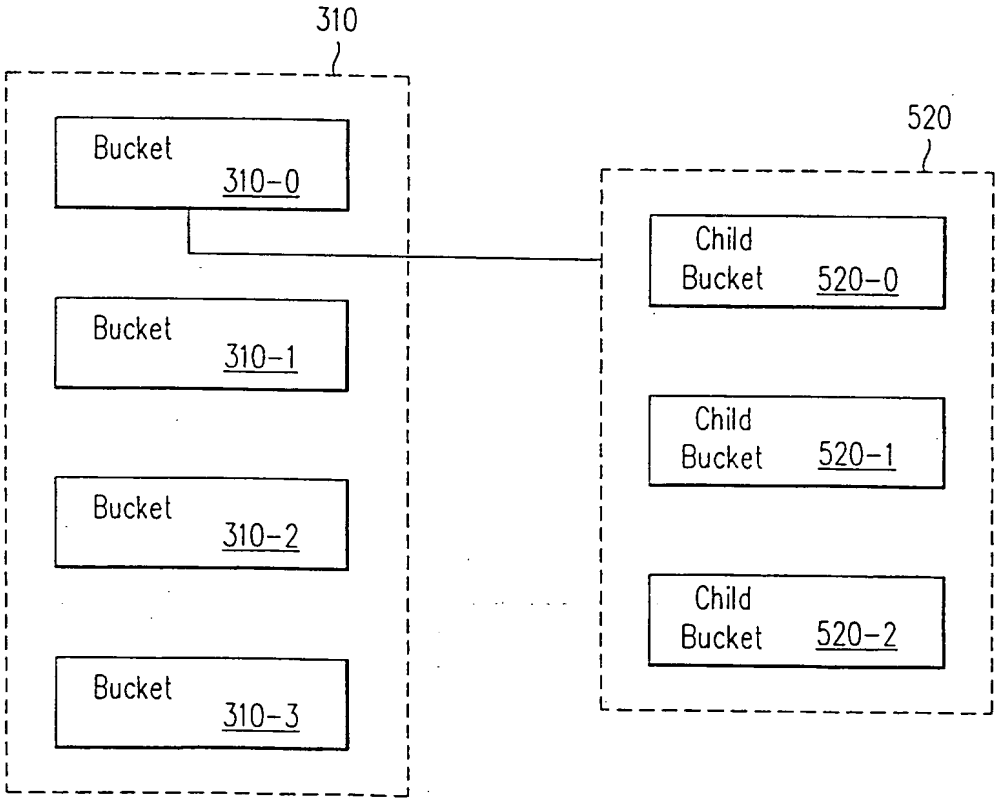
310
Bucket 310-0
Bucket 310-1
Bucket 310-2
Bucket 310-3
520
Child Bucket 520-0
Child Bucket 520-1
Child Bucket 520-2

FIG. 5B

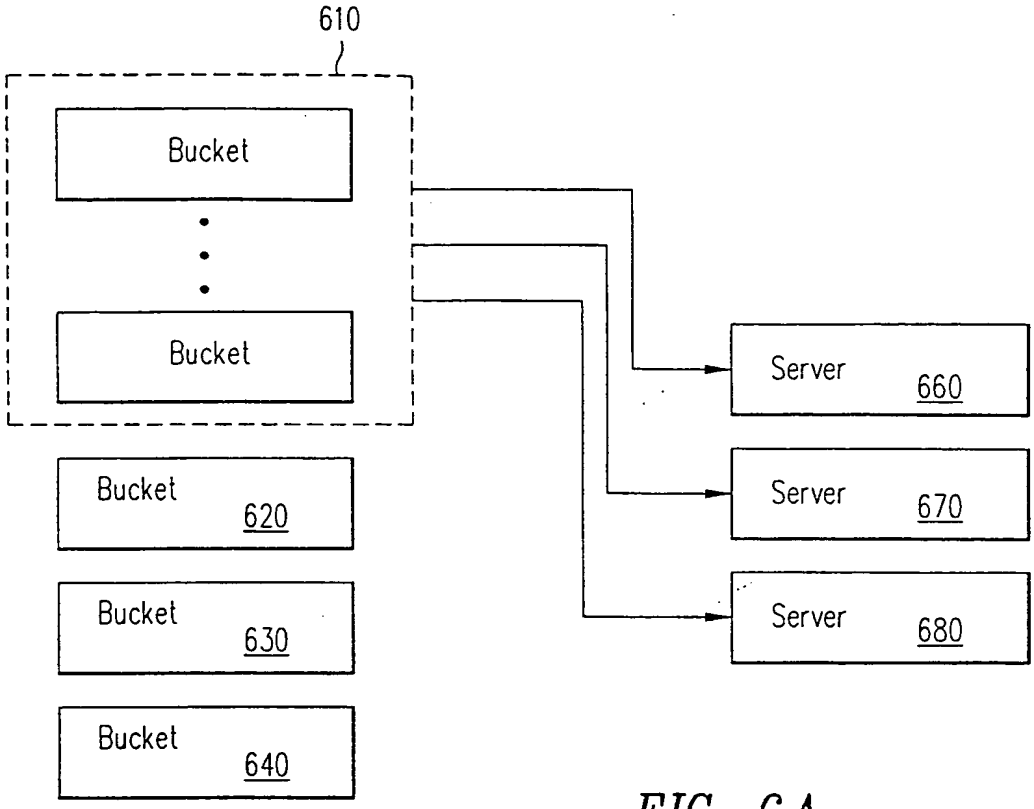
610
Bucket
Bucket
Bucket 620
Bucket 630
Bucket 640
Server 660
Server 670
Server 680

FIG. 6A

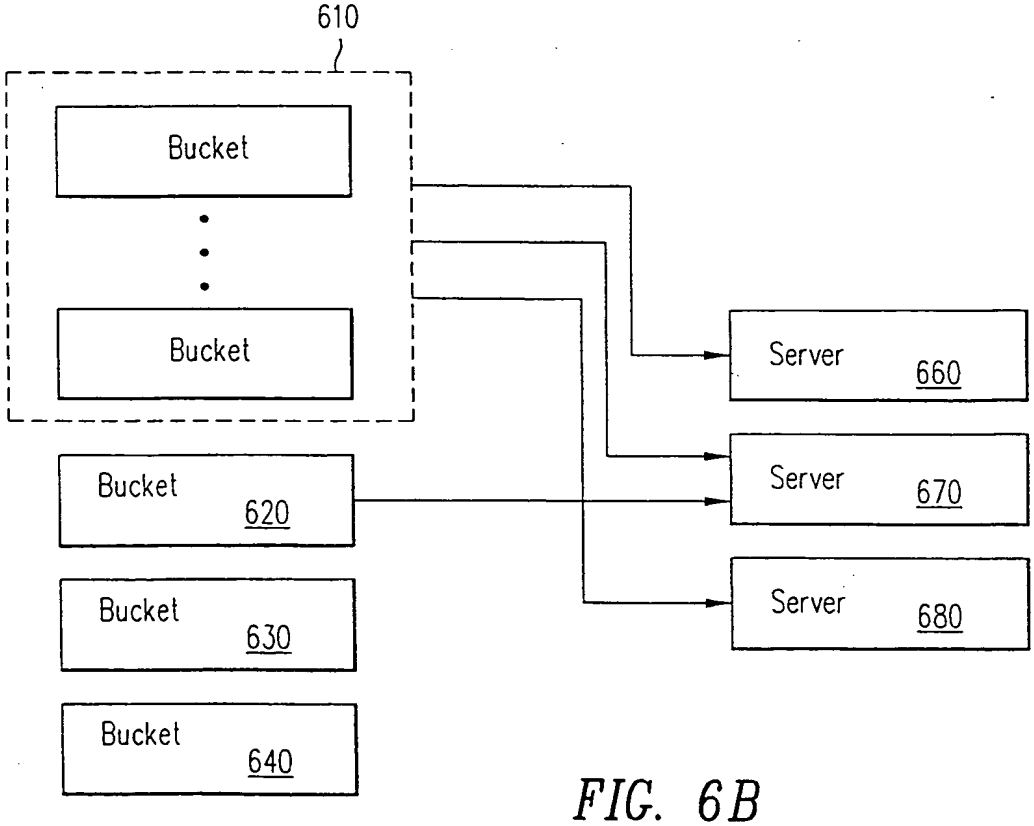
610
Bucket
Bucket
Bucket 620
Bucket 630
Bucket 640
Server 660
Server 670
Server 680

FIG. 6B

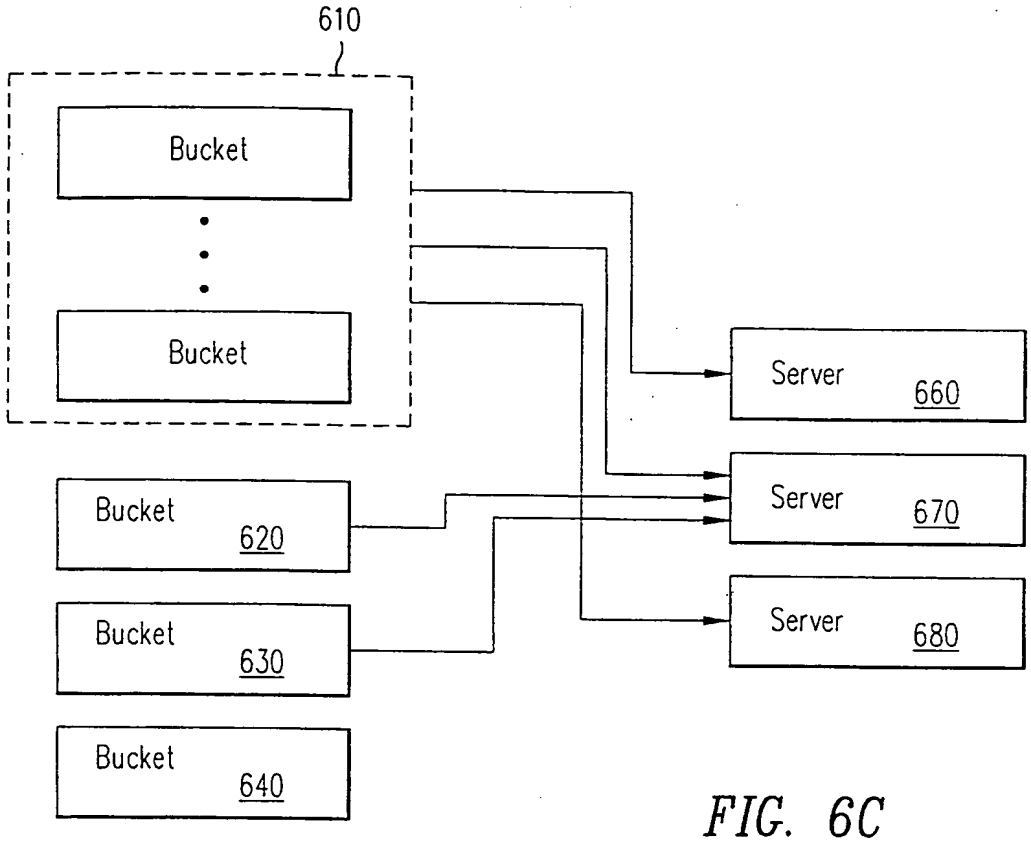
610
Bucket
Bucket
Bucket 620
Bucket 630
Bucket 640
Server 660
Server 670
Server 680

FIG. 6C

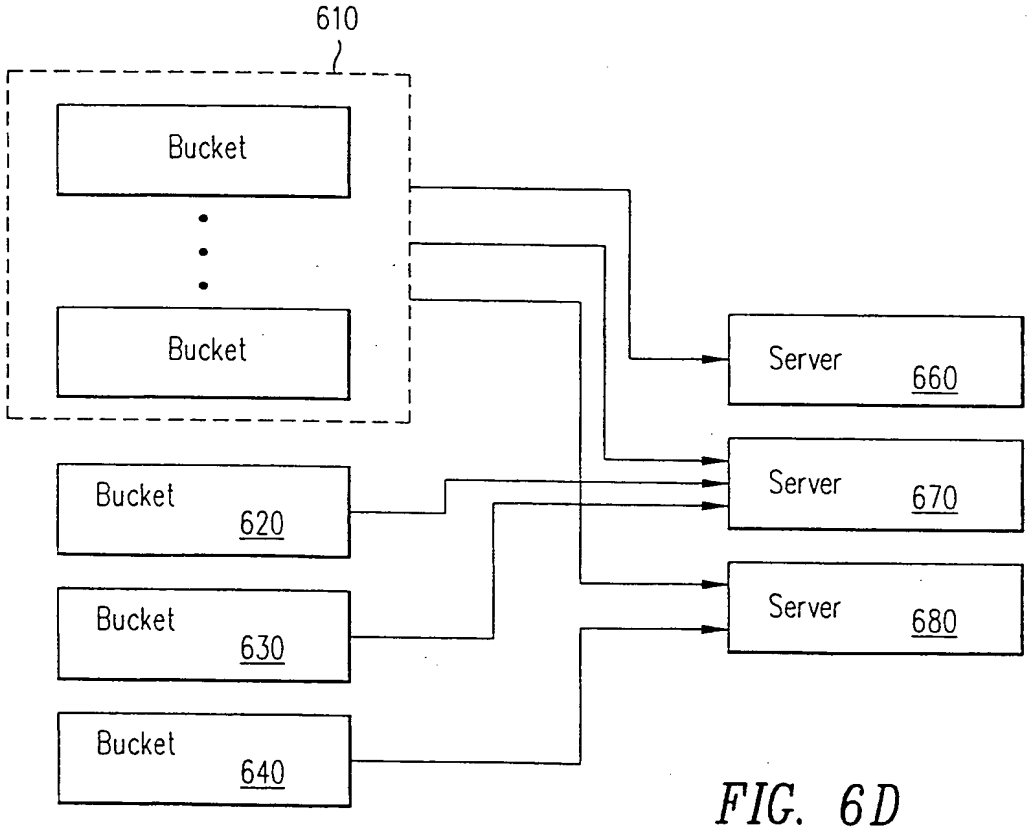
610
Bucket
Bucket
Bucket 620
Bucket 630
Bucket 640
Server 660
Server 670
Server 680

FIG. 6D

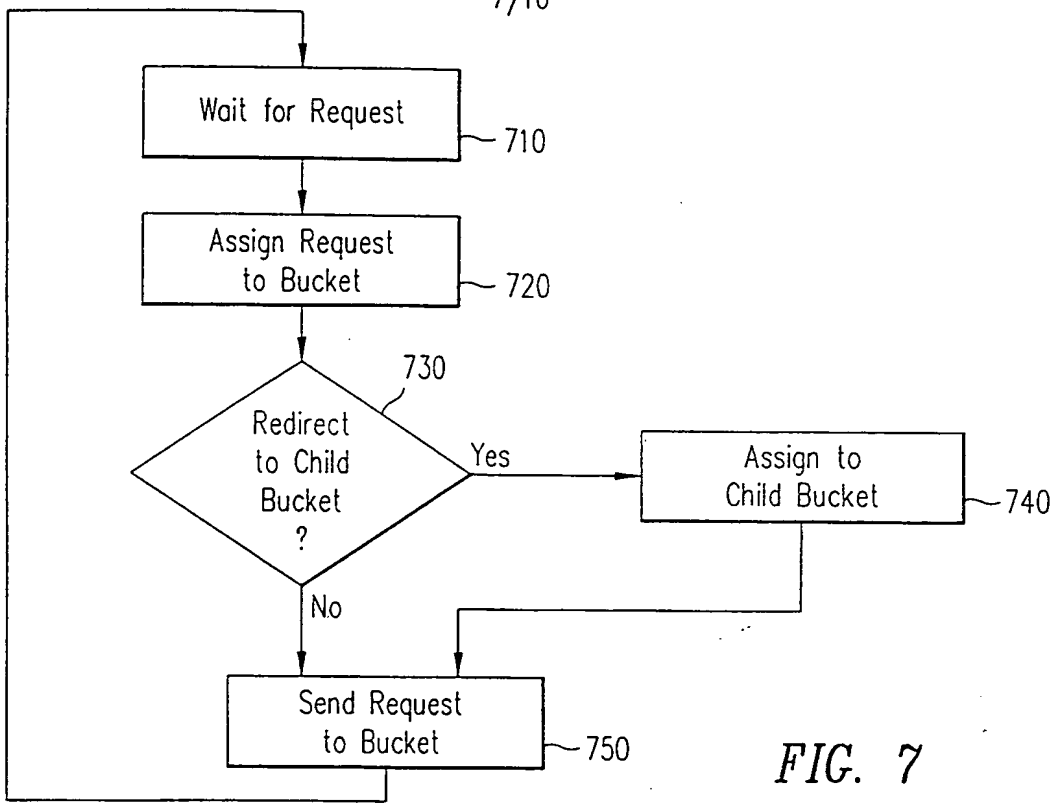
Wait for Request
710
Assign Request to Bucket
720
730
Redirect to Child Bucket ?
Yes
Assign to Child Bucket
740
No
Send Request to Bucket
750

FIG. 7

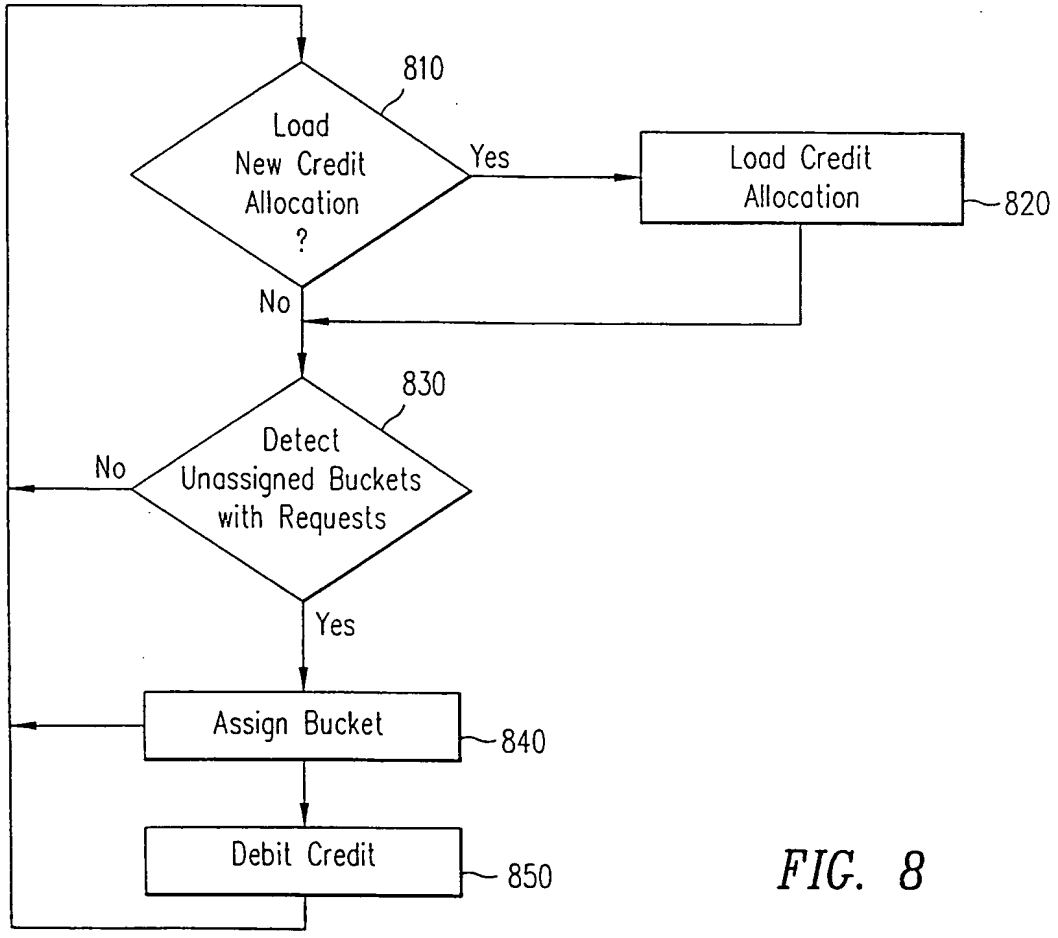
810
Load New Credit Allocation ?
Yes
Load Credit Allocation
820
No
830
Detect Unassigned Buckets with Requests
No
Yes
Assign Bucket
840
Debit Credit
850

FIG. 8

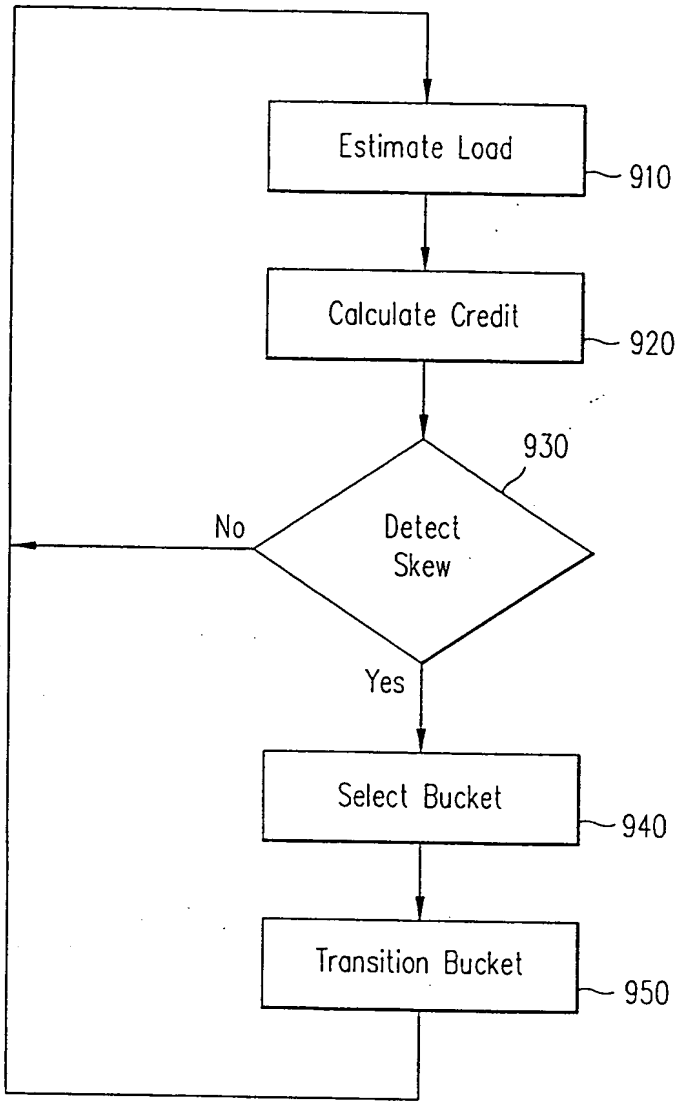
Estimate Load
910
Calculate Credit
920
930
Detect Skew
No
Yes
Select Bucket
940
Transition Bucket
950

FIG. 9

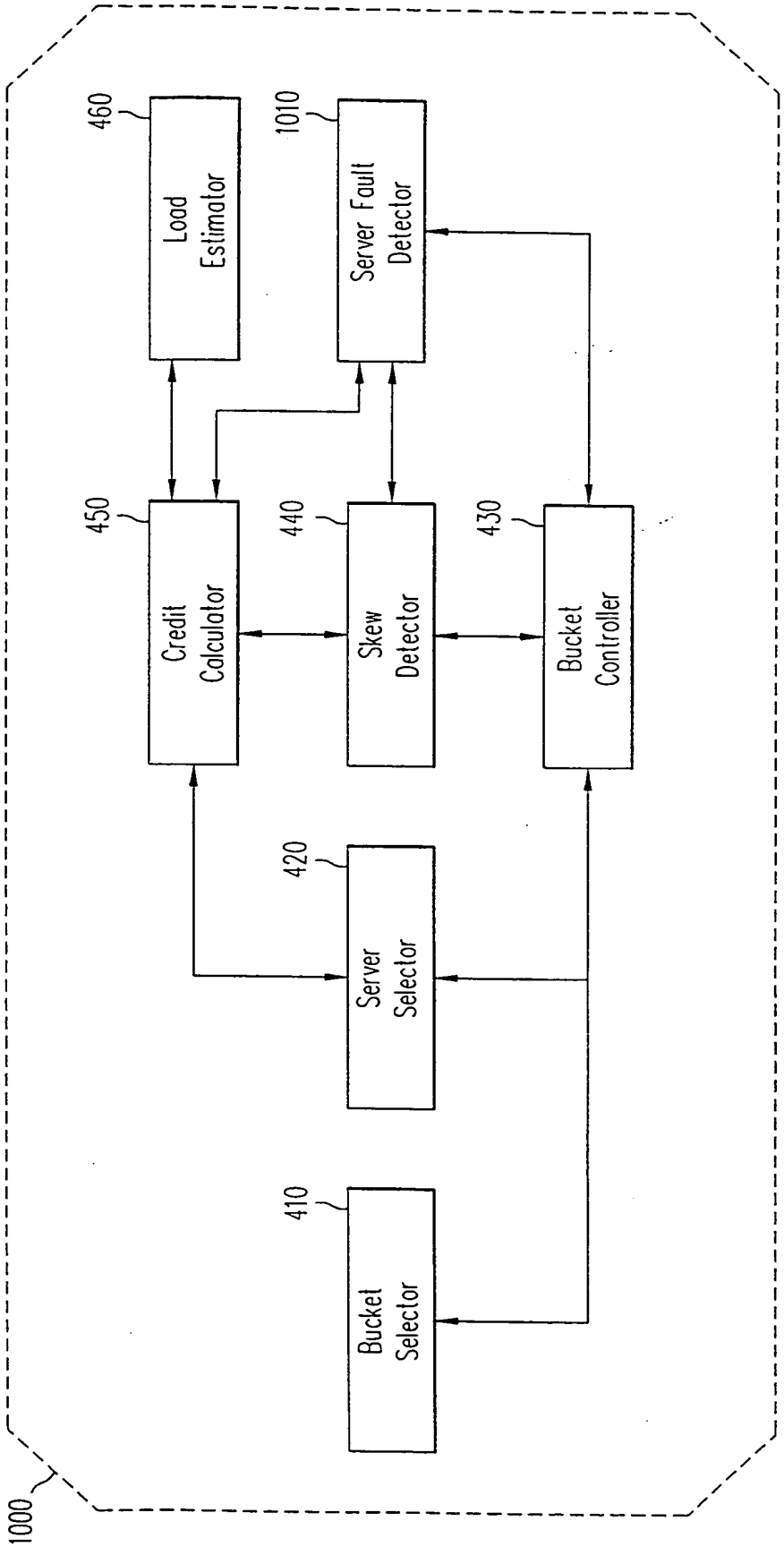
460
Load Estimator
1010
Server Fault Detector
450
Credit Calculator
440
Skew Detector
430
Bucket Controller
420
Server Selector
410
Bucket Selector
1000

FIG. 10

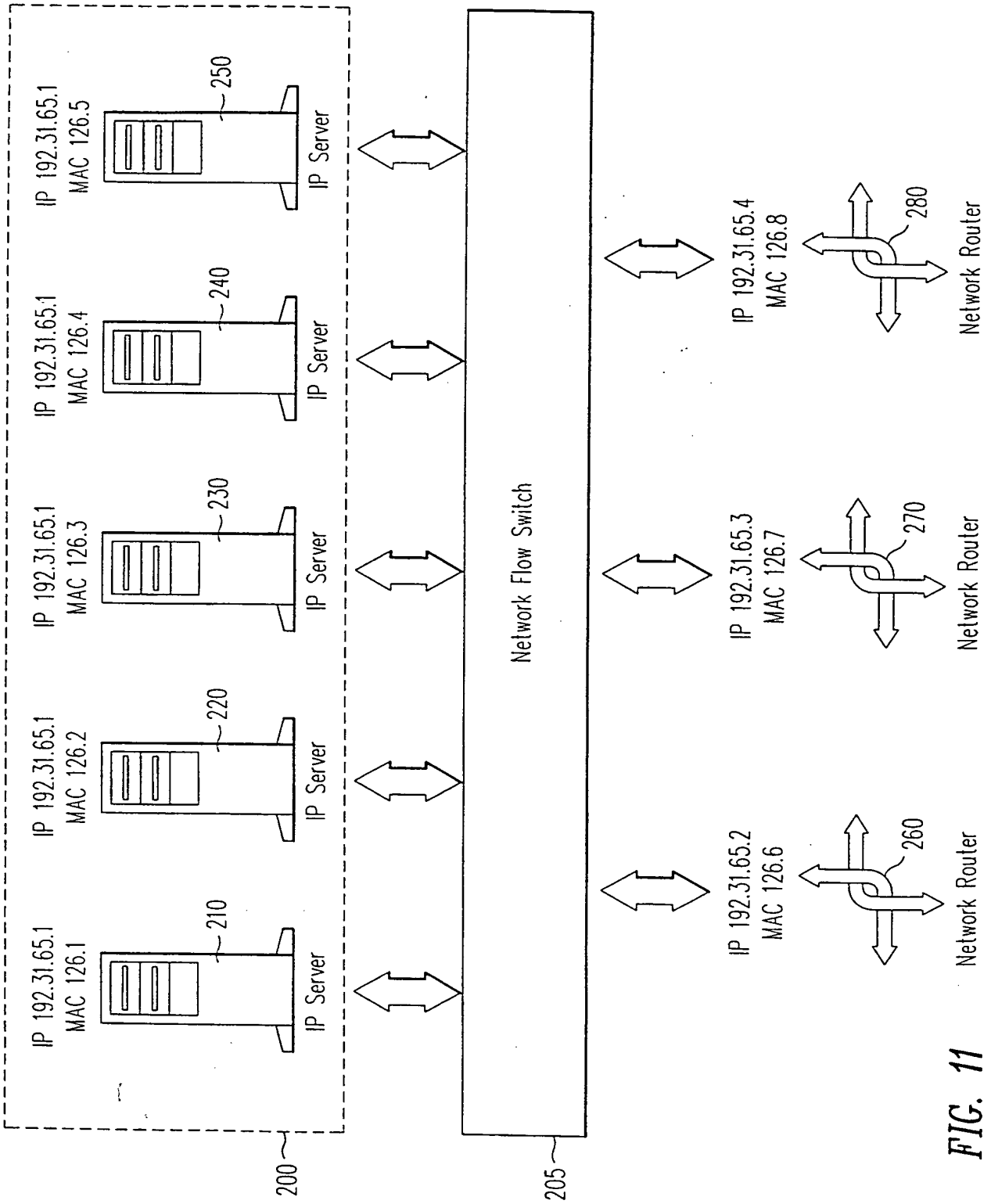
IP 192.31.65.1
MAC 126.1
210
IP Server
IP 192.31.65.1
MAC 126.2
220
IP Server
IP 192.31.65.1
MAC 126.3
230
IP Server
IP 192.31.65.1
MAC 126.4
240
IP Server
IP 192.31.65.1
MAC 126.5
250
IP Server
200
Network Flow Switch
205
IP 192.31.65.2
MAC 126.6
260
Network Router
IP 192.31.65.3
MAC 126.7
270
Network Router
IP 192.31.65.4
MAC 126.8
280
Network Router

FIG. 11